Energy Minimization for Federated Asynchronous Learning on Battery-Powered Mobile Devices via Application Co-running

Cong Wang†*, Bin Hu†, and Hongyi Wu†
†George Mason University, Fairfax, VA 22030, USA
§Old Dominion University, Norfolk, VA 23529, USA
†cwang51@gmu.edu
§bhu@odu.edu, h1wu@odu.edu

Abstract—Energy is an essential, but often forgotten aspect in large-scale federated systems. As most of the research focuses on tackling computational and statistical heterogeneity from the machine learning algorithms, the impact on the mobile system still remains unclear. In this paper, we design and implement an online optimization framework by connecting asynchronous execution of federated training with application co-running to minimize energy consumption on battery-powered mobile devices. From a series of experiments, we find that co-running the training process in the background with foreground applications gives the system a deep energy discount with negligible performance slowdown. Based on these results, we first study an offline problem assuming all the future occurrences of applications are available, and propose a dynamic programming-based algorithm. Then we propose an online algorithm using the Lyapunov framework to explore the solution space via the energy-staleness trade-off. The extensive experiments demonstrate that the online optimization framework can save over 60% energy with 3 times faster convergence speed compared to the previous schemes.

Keywords—Asynchronous federated learning; on-device deep learning; energy-efficiency; power-aware online optimization.

I. INTRODUCTION

Our planet is in danger. Among a variety of causes, AI plays an unequivocally negative role to accelerate the emission of carbon dioxide and irreversible climate change. As deep learning is increasingly deployed in large-scale distributed systems, their energy footprint is growing at an unprecedented, breathtaking rate [1]. Recently, Federated Learning rises as a promising computing paradigm that allows participants to learn a collaborative model in privacy-preserved manner [2]–[5]. However, by pushing neural computations to the multi-core CPUs [7], its energy implication is far from clear on battery-powered mobile devices [8]: high-intensity neural computation quickly drains the battery and frequent charge/discharge also shorten the battery lifetime, and their disposal ultimately becomes an environmental liability.

The classic federated learning originates from the principles of synchronous Stochastic Gradient Descent (Sync-SGD) in cloud computing, where all the participants proceed in lock-step and their parameters are averaged at the parameter server. It is well-known that such simple migration is subject to the computational heterogeneity in mobile environments, vastly due to the segmented mobile hardware market and vendor-supplied drivers. Worst-case stragglers (slowest workers) could be orders of magnitude slower than the average execution whereas the majority of the computing power is underutilized, especially when the stragglers are experiencing heavy thermal throttling and user interference [6]. Further, Sync-SGD does not provide the temporal flexibility of coordination and slow convergence further exacerbates the energy consumptions in the system. Asynchronous training (ASync-SGD) is a competitive solution to tackle computational heterogeneity [9] but its potential is yet to be fully explored in federated learning [13]. ASync-SGD allows fast participants to proceed in lock-free steps while the global parameters are exchanged and kept with the most updated local ones. Without such barrier from the stragglers, more updates can be made and the wall-clock convergence time is reduced. Unfortunately, most of the research in Sync-SGD [2]–[5] and ASync-SGD [9]–[11], [13] lie in the confined areas of machine learning and optimization theories, but the interaction to the underlying system is not fully explored, particularly for achieving energy-efficient computation.

In this paper, we aim to optimize the energy expenditure of federated learning tasks by taking advantage of application co-running opportunities and asynchronous execution. The design stems from the pervasive ARM CPU microarchitecture, which features the big.LITTLE cores [14] to tackle multi-tasking with energy-efficiency: the big cores of high throughput for foreground applications and the little cores of low power consumptions for system and background processes. To avoid interfering with the normal usage, the training threads can be designated as a background service and only called once a set of conditions such as networking, battery energy conditions are met. As validated in our experiments, once the training threads of high parallelization are running in the background (dispatched to the little cores by the kernel scheduler), simultaneous execution of a foreground application gives the entire system a deep energy discount (about 30-50%) compared to running the foreground application and training separately, with negligible performance impact measured by Frame per Second (FPS). Combined with ASync-SGD, it gives the flexibility to defer gradient updates until a foreground application takes place.

A seamless integration of the system dynamics with the upper-level machine learning algorithm faces tremendous challenges. The success of asynchronous training relies on well-managed staleness in the system, that the stale updates
A. Asynchronous Federated Learning

subject to extensive heterogeneity in mobile edge systems [2]. As pointed out in [3]–[5], this simple migration is with a barrier until all the workers finish their local training. This work establishes on synchronous SGD, where local workers proceed with a barrier until all the workers finish their local training [2]. As pointed out in [3]–[5], this simple migration is subject to extensive heterogeneity in mobile edge systems due to the diverse computational capability, network connectivity/bandwidth and user behaviors. [3] develops a new aggregation rule to allow local variations such as the number of epochs and optimizers used by different participants. [5] analyzes the convergence instability due to stragglers and proposes a mechanism to correct those diverging gradient updates. [4] adds a proximal term to the objective to manage heterogeneity associated with partial information, when the straggler’s updates have been dropped out. These works aim to improve the computational efficiency of Sync-SGD while preserving the statistical stability.

ASYNC-SGD is a natural way to tackle computational heterogeneity, and its original version can be traced back to HOGWILD! [9] in multicore systems. Multiple threads are allowed access to the shared memory and update the model at will. Considerable efforts have been devoted to understanding and mitigating staleness [10], [11], [10] adopts the Taylor Expansion and Hessian approximation to compensate the delay from stale gradients, while avoiding the complexities from the high-order terms. [11] introduces a regularized term to reduce the variance due to staleness. Though asynchrony introduces race conditions, it is proved to achieve optimal convergence rate at a much faster speed [9], mainly due to more number of updates are now being conducted. Some works also partially contribute the statistical efficiency to the implicit momentum introduced from the stale gradients [12].

Momentum plays an important role to facilitate convergence. The update is simply an exponentially weighted average that continuously adds a portion of the previous momentum vector \( v_{t-1} \) to the current vector \( v_t \) plus the fraction from the current gradient vector \( s_t \),

\[
v_t = \beta v_{t-1} + (1 - \beta) s_t, \quad \theta_t = \theta_{t-1} - \eta v_t \quad (1)
\]

then the model parameters \( \theta_t \) are updated according to the learning rate \( \eta \). Here, the stale gradients can be thought as the previous gradient vectors \( v_{t-1} \) that can dampen oscillations along the way to the minima. Thus, it is interesting to see that the benefits and drawbacks of staleness actually co-exist, but such contradiction and its theoretical implication are still not fully understood at this stage [13]. Most of the federated research aims at improving the Sync-SGD or ASync-SGD algorithms, but overlooks important aspects from the system, such as energy-efficiency on battery-powered edge devices. This work differs from a large body of existing works to combine ASync-SGD with system-level opportunities and reduce energy footprint in federated systems.

B. Energy Optimization

The efforts of energy optimization on mobile devices has been revolving around software and hardware components to elongate battery lifetime, e.g., dynamic voltage and frequency scaling, resolving “energy bugs” from unexpected energy consumption [16] and coalescing packets to reduce tail energy on the wireless network interface [17], [18] using the Lyapunov framework [27]. For on-device training [7], delegating the long-running, training workloads as a background service is

II. BACKGROUND AND RELATED WORKS

A. Asynchronous Federated Learning

Sync-SGD. The state-of-the-art Federated Learning establishes on synchronous SGD, where local workers proceed with a barrier until all the workers finish their local training [2]. As pointed out in [3]–[5], this simple migration is subject to extensive heterogeneity in mobile edge systems. The state-of-the-art Federated Learning establishes on synchronous SGD, where local workers proceed with a barrier until all the workers finish their local training [2]. As pointed out in [3]–[5], this simple migration is subject to extensive heterogeneity in mobile edge systems. The contributions of this work are many-fold. First, motivated by a series of key findings in real experiments, we leverage ASync-SGD and system-level opportunities for energy optimization of federated training tasks on edge devices. To the best of our knowledge, this is one of the few works that integrate the high-level machine learning algorithms with the low-level system dynamics on consumer’s edge devices. Second, we formulate both offline and online optimization problems and design an efficient online scheduler while ensuring bounded staleness in the long term. Finally, we conduct extensive evaluations on a mobile testbed with 4 types of devices using the CIFAR10 dataset [33]. The results demonstrate over 60% energy saving compared with FedAvg [2] and 3× faster convergence speed.

The rest of the paper is organized as follows. Section II discusses the background and related works. Section III motivates this work and defines the system model. Sections IV and V describe the framework for offline and online optimization. Section VI implements the framework on edge devices. Section VII evaluates the proposed framework and Section VIII concludes this work with future directions.
a viable way to avoid interrupting normal usage. However, its performance is still unclear since the mobile systems are built with event-driven, user-centric designs to render the best performance for the foreground applications. Fortunately, the big.LITTLE architecture extends the capacity to handle concurrent low-intensity workloads on the more energy-efficient cores [14]. Since a running foreground application would have already activated shared resources on the big cores and the background processes are typically dispatched to the little cores, co-running training with applications could take advantages of such energy disproportionality [19]. Similar to packet coalescing [17], [18], this idea of task bundling dates back to piggyback sensing activities with applications such as web browsing and phone calls [20]. However, these early works cannot be readily applied to federated learning to achieve the energy-staleness trade-off as well as bounded staleness with statistical stability. The closest works to ours are [17], [18] that adopt the Lyapunov framework to achieve energy-delay trade-offs. This paper takes a step forward to consider gradient staleness induced by delayed execution and attempts to fill the gap between the machine learning algorithms and mobile systems for optimal energy efficiency.

III. MOTIVATION AND SYSTEM MODEL

A. Preliminary Experiment

We motivate the design by conducting some preliminary power measurements on the HiKey 970 Development Board [21] and Pixel2 smartphone (see Sec. VI for implementation details). Assuming the user is going to run an application at a certain time, we compare the power consumption of two approaches: 1) schedule training as a service in the background, independently from the upcoming application (separate). 2) schedule the training task to execute together with the foreground application (co-running) and the application also stops when training finishes. Since applications have diverse resource demands and patterns of user interactions, we choose some popular applications from Google Play as shown in Fig. 1. To verify that co-running does not lead to noticeable slowdown, we also perform some experiments on Pixel2 to see the rendering effects perceived by the user as measured in Frame Per Second (FPS) in Fig. 2. The important observations are summarized below.

Observations 1. Compared to separate scheduling, co-running offers 35-50% energy saving. We notice that the little cores designated for executing the training task typically have 95-98% utilization whereas the big cores have 30-50% utilization depending on the foreground application.

The energy saving originates from the asymmetric CPU microarchitecture. Though the big/little core clusters have their own cache hierarchies, the memory bandwidth is shared. If the memory resource is already activated by the highly paralleled neural operations on the little cores and kept at certain power state, foreground applications executed on the big cores should not elevate the power state too much on the shared resources. Thus, co-running typically offers a substantial energy saving compared to separate executions. This is also validated by more experiments with the homogeneous cores in Nexus 6 device as resource contention on the same cluster degrades the energy saving percentage (see more details in Sec. VII).

Observations 2. Co-running might lead to slowdown to the training tasks depending on the intensity of the foreground applications. For some lightweight applications such as news and web browsing, the training task does not exhibit any slowdown; for intensive applications such as gaming, we notice about 10-15% slowdown due to resource contention since a higher priority is given to the foreground applications by design. However, co-running still provides an overall energy saving despite of slightly elongated execution time.

Observations 3. Co-running does not have noticeable slowdown for the foreground applications, as the average FPS stays steadily around 60 and 30 frames/s shown in Fig. 2.

B. System Model

A device pulls the current model from the parameter server when it becomes available depending on the network condition or battery energy. Training is either immediately scheduled or postponed until an application co-running opportunity. If the decision is co-running, the power consumption is $P_{a_i}$ on the i-th device; otherwise, separate executions of training and application take $P_{b_i}$ and $P_{a_i}$ respectively. The execution time of training is $d_i$ at the i-th user. For simplicity, it is assumed that the application would last for the same time duration of the training task. After the local epoch is

\[ P_{a_i} \]

1The power consumption of training is stable as the CPU typically stays at the maximum frequency during training. For applications, the power consumption fluctuates due to user interaction and frequency scaling. Thus, we measure the average power consumption as shown in Table II.
finished, the model is pushed to the server to update the global parameters and ready to be downloaded by other participants in the following time slots. We formally define the lag and gradient gap to quantify gradient staleness.

**Definition 1. (Lag)** The lag \( l_\tau \) is defined as the number of updates from other users that have been made to the global model within the time interval \([t, t + \tau]\). \( t \) is the initial time when the user receives the model from the server and \( t + \tau \) is the time that the user finishes training and applies the parameters to the global model.

Sync-SGD guarantees the gradient aggregations are aligned in lock-step so \( l_\tau = 0 \). For ASync-SGD, Fig.3 shows an example of three users \( i, j, k \) and the control decision for \( i \) is co-running at time \( t^a \) when the application arrives. Users \( j \) and \( k \) immediately perform training without waiting for the applications and finish before \( t + \tau \). Therefore, during the time interval of \([t, t + \tau]\), there are \( l_\tau = 2 \) updates made from other users \( j \) and \( k \) to the global model, whereas the model update at \( t + \tau \) is computed from a stale version at time \( t \). Using the metric of lag alone cannot precisely quantify the difference between the two updates. Thus, we introduce another definition.

**Definition 2. (Gradient Gap)** The gradient gap \( g(t, t + \tau) \) can be calculated by the norm difference of the parameters \( \theta_t \) and \( \theta_{t+\tau} \) [30], [31],

\[
g(t, t + \tau) = \|\theta_{t+\tau} - \theta_t\|_2. \tag{2}
\]

We adopt the efficient Linear Weight Prediction [32] to estimate the global parameter \( \theta_{t+\tau} \) in the future time \( t + \tau \),

\[
\theta_{t+\tau} = \theta_t - \eta \frac{1 - \beta^\tau}{1 - \beta} v_t. \tag{3}
\]

\( \eta \) is the learning rate. \( \beta \) and \( v_t \) are the momentum coefficient and vector defined in Eq. (1). Plugging Eq. (3) into Eq. (2), we have

\[
g(t, t + \tau) = \|\frac{1 - \beta^\tau}{\eta (1 - \beta)} v_t\|_2. \tag{4}
\]

**IV. Offline Scheduling Problem**

In this section, we first study an offline solution assuming that all application occurrences are known, which serves as an optimal solution and baseline for the online algorithm proposed next.

**Problem Formulation.** Energy optimization aims to achieve two conflicting goals to maximize the energy saving and avoid staleness. Given the application arrivals, the goal is to maximize the sum of energy saving from all the users: decide whether to co-run training with application for each user. Denote the number of users by \( n \). For the \( i \)-th device, the energy saving \( s_i = P_i^b + P_i^c - P_i^a \) if the decision is to co-run with application (decision variable \( x_i = 1 \)); otherwise the energy saving is 0 (\( x_i = 0 \)).

\[
P1: \quad \max \sum_{i=1}^{n} s_i x_i \tag{5}
\]

subject to

\[
\sum_{i=1}^{n} g_i(t_i, t_i + \tau) x_i \leq L_b, \tag{6}
\]

\[
x_i \in \{0, 1\}. \tag{7}
\]

Constraint (6) imposes that the sum of gradient gaps is bounded by \( L_b \) and (7) makes \( x_i \) 0-1 valued. This can be considered as a Knapsack Problem [26], which maximizes the total value of items under a weight capacity and our problem maximizes the energy saving under the staleness bound \( L_b \). Since the problem is NP-complete, it can be efficiently solved by utilizing the optimal sub-structure with dynamic programming. The equation of the maximal energy saving \( S_i(y) \) is,

\[
S_i(y) = \begin{cases} 
S_{i-1}(y), & 0 < y \leq g_i(t_i, t_i + \tau) \\
\max \left\{ S_{i-1}(y), S_{i-1}(y - g_i(t_i, t_i + \tau)) + s_i \right\}, & \text{otherwise}
\end{cases} \tag{8}
\]

A key difference from the original Knapsack solution is that \( g_i(t_i, t_i + \tau) \) is computed based on the lag \( l_\tau \), which in turn, depends on the decisions of other users - this creates a looping situation. We know that, in the worst case, the lag \( l_\tau \) is bounded by \( n - 1 \) because the rest of devices could have all made their updates within \( \tau \). To tighten this, we further reduce this value as described in the next lemma. This is because given all the beginning time, application arrival and training duration, for each device, some of the rest devices should be out of the training interval and do not count towards the lag. This allows us to obtain a tighter upper bound on \( l_\tau \) without knowing the control decisions in advance. As long as this upper bound is within \( L_b \), we have a feasible, sub-optimal solution.

**Lemma 1.** Given the beginning time \( t_i \), application arrival time \( t_i^a \) and duration of training \( d_i \) for user \( i \), the lag for \( i \) is bounded by,

\[
l_\tau \leq \sum_{j=1}^{n-1} \begin{cases} 
1(t_j^a + d_j + t_j^a) \vee [t_j^a, t_j^a + d_j] \\
1(\{t_j + d_j + t_j^a + d_j\} \vee [t_j^a, t_j^a + d_j])
\end{cases}, \tag{9}
\]

in which \( \vee \) denotes the logical “or” of the two time intervals and \( 1(\cdot) \) is one if the training ends in one of the time intervals.

**Proof:** It can be proved by considering all possible decisions for each pair of \( i \) and \( j \) \( \{1, \cdots, n - 1\} \). For \( i \), it has two scheduling possibilities: 1) execute training at \( t_i \) and end at \( t_i + d_i \) (the interval of \([t_i, t_i + d_i]\)); 2) co-run training with application arrival at \( t_i^a \) and end at \( t_i^a + d_i \) (the interval of \([t_i^a, t_i^a + d_i]\)). Meanwhile, any other \( j \) has the similar possibilities to end at \( t_j + d_j \) or \( t_j^a + d_j \). If any of
Algorithm 1: Offline Algorithm

1. Input: app. arrival time \( t_i \), training execution \( d_i \), ∀ \( i \)
2. Output: scheduling decisions \( x_i \) and maximum energy saving.
3. Initialize \( S_0(y) = 0, y \geq 0 \).
4. for \( i = 1 \) to \( n \) do
5.   for \( j = 1 \) to \( L_b \) do
6.     if \( y \leq g_i(t_i, t_i + \tau_i) \) then
7.       \( S_i(y) \leftarrow S_i(y) \).
8.     else
9.       \( S_i(y) \leftarrow \max \{ S_i(y), S_i(y - g_i(t_i, t_i + \tau_i)) + s_i \} \).

these intervals from \( i \) and \( j \) has overlaps, the gradient gap is increased by one and summed over all \( n - 1 \) devices.

Based on Lemma 1, the offline solution is summarized in Algorithm 1 with a time complexity of \( \mathcal{O}(nL_b) \), and will serve as a baseline for the online algorithm discussed next.

V. ONLINE SCHEDULING

Offline scheduling assumes the future application arrival as a priori. In this section, we propose an online scheduling with the Lyapunov framework that only relies on the current observation. We consider a task queue for the entire system as defined below.

Definition 3. (Queue Dynamics). The task queue represents the number of users waiting to be scheduled. The arrival of users can be considered as a random process \( A(t) \). The queue backlog will increase by \( A(t) - n \) if a number of \( n \) users are ready to start training at \( t \). If \( m \) users finish their training in a time slot, the queue backlog is reduced by \( b(t) = m \).

Assume time is equally slotted with the length of \( t_d \). The system makes a control decision \( \alpha(t) \) at time \( t \). The energy consumption \( P_i(t) \) of the \( i \)-th device depends on how training is scheduled and the current application status \( s(t) = \{\text{app}, \text{no app}\} \), i.e., \( P_i(t) = P_i(\alpha(t), s(t)) \):

\[
P_i(t) = \begin{cases} 
P_i^n t_d, & \alpha(t) = \text{schedule}, s(t) = \text{app} \smallskip 
P_i^n t_d, & \alpha(t) = \text{schedule}, s(t) = \text{no app} \smallskip 
P_i^d t_d, & \alpha(t) = \text{idle}, s(t) = \text{app} \smallskip 
P_i^d t_d, & \alpha(t) = \text{idle}, s(t) = \text{no app}. \end{cases} \quad (10)
\]

According to the experimental measurements, \( P_i^n > P_i^a > P_i^d > P_i^n \). The corresponding service rate for \( i \) is:

\[
b_i(t) = \begin{cases} 
1, & \alpha(t) = \text{schedule} \smallskip 
0, & \alpha(t) = \text{idle}. \end{cases} \quad (11)
\]

and the service rate in the system is \( b(t) = \sum_{i=0}^{n} b_i(t) \). The gradient gap is,

\[
g_i(t, t + \tau_i) = \begin{cases} 
\eta \frac{1 - \beta^d}{1 - \beta} v_i, & \alpha(t) = \text{schedule} \smallskip 
g_i(t - 1, t + \tau_i - 1) + \epsilon, & \alpha(t) = \text{idle} \end{cases} \quad (12)
\]

To simplify the analysis, we take an approximation here to make the actual service and deduction of queue length at \( t + d_i \) to be effective at \( t \).

If the decision is to schedule training, the gap is computed using Eq. (4) with lag \( l_d \) during the execution time of \( d_i \); if the decision is to remain idle, the gap is cumulative from the previous slot plus a small time-averaged gap increment \( \epsilon \), which estimates the impact on the gradient gap for each idle time slot. The sum of gradient gaps is \( G(t, t + \tau) = \sum_{i=0}^{n} g_i(t, t + \tau) \).

Problem Formulation. Our goal is to minimize the time-averaged energy consumption of training tasks in the system of \( n \) users,

\[
P_2: \quad \limsup_{T \to \infty} \frac{1}{T} \sum_{i=1}^{T} E\{P_i(t)\} \quad (13)
\]

s.t.

\[
\limsup_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} g_i(t, t + \tau) \leq L_b \quad (14)
\]

Eq. (14) guarantees that the sum of gradient gaps from all the participants is bounded in a time averaged sense. \( P_2 \) can be transformed into the queue stability problem under the Lyapunov optimization framework. Given the arrival rate \( A(t) \) and service rate \( b(t) \), the queueing dynamics is,

\[
Q(t + 1) = \max (Q(t) - b(t), 0) + A(t) \quad (15)
\]

with the initial \( Q(0) = 0 \). We define a virtual queue \( H(t) \) for constraint (14),

\[
H(t + 1) = \max (H(t) + \sum_{i=1}^{n} g_i(t, t + \tau) - L_b, 0) \quad (16)
\]

and the initial \( H(0) = 0 \). We concatenate the actual and virtual queues into \( \Theta(t) = [Q(t), H(t)] \), define the Lyapunov function \( L(\Theta(t)) \) as the queue congestion of the backlogged training tasks,

\[
L(\Theta(t)) \triangleq \frac{1}{2} (Q(t)^2 + H(t)^2) \quad (17)
\]

and the Lyapunov drift function \( \Delta(\Theta(t)) \) as:

\[
\Delta(\Theta(t)) \triangleq \mathbb{E}\{L(\Theta(t+1)) - L(\Theta(t))|\Theta(t)\} \quad (18)
\]

It represents the change in the Lyapunov function in time slot \( t \) representing the scalar volume of queue congestions. The new optimization problem is to minimize the drift-plus-penalty:

\[
P_3: \quad \min \Delta(\Theta(t)) + V\mathbb{E}\{P(t)|\Theta(t)\} \quad (19)
\]

\( V \) is the control parameter to balance between energy and staleness. Following the Lyapunov framework, the key is to obtain the upper bound of the drift as described in the following Lemma.

Lemma 2: Given the queue backlogs \( Q(t) \), arrival rates \( A(t) \) and service rate \( b(t) \) and the gradient gaps, we have the following upper bound for the drift-plus-penalty term,

\[
\Delta(\Theta(t)) + V\mathbb{E}\{P(t)|\Theta(t)\} \leq B + V\mathbb{E}\{P(t)|\Theta(t)\} + Q(t)\mathbb{E}\{(A(t) - b(t)|\Theta(t)) + H(t)\mathbb{E}\{G(t, t + \tau) - L_b|\Theta(t)\}\} \quad (20)
\]
where $B = \frac{1}{2} \left( A_{max}^2 + B_{max}^2 + G_{max}^2 + L_0^2 \right)$ is a positive constant. The proof can be found in Appendix X.

Our algorithm observes the current queue backlogs of $Q(t)$, $H(t)$ and application status $s(t)$ to make a decision of $\alpha(t) \triangleq \{ \text{‘schedule’, ‘idle’} \}$ that minimizes the R.H.S. of the drift bound Eq. (20), which is equivalent to the objective in Eq. (19).

$$\min \left( V \sum_{i=1}^{n} P_i(t) - Q(t) \sum_{i=1}^{n} b_i(t) + H(t) \sum_{i=1}^{n} g_i(t, t + \tau_i) \right)$$

(21)

This formulation makes online decisions based on the current observations and does not need a-priori knowledge of the arrival rates. With the information of application usage, a centralized implementation can be conducted in $O(n)$ at the parameter server. However, since application usage are considered as private and their patterns can be used to re-identify specific users [28], centralization carries certain privacy risks.

A. Distributed Implementation

The minimization of Eq. (21) can be achieved in a distributed manner via appropriate information exchange between the server and the users. We design distributed implementations into the Lyapunov framework that can mitigate the privacy leakage of application usage to the parameter server. In time $t$, each user minimizes Eq. (21) from the control decision space based on status of application usage and queue backlogs, thus the application usage $s_i(t)$ is not leaked to the server. In the last term of Eq. (21), the user computes the gradient gap $g_i(t, t + \tau_i)$ according to Eq. (2). If the decision is “schedule”, the number of updates in the time interval of $[t, t + \tau]$ can be supplied by the server with the estimated arrival time of the running tasks; otherwise, the gap accumulates from the previous value plus a small increment according to Eq. (12). Hence, for each user $i$, the decision making fully depends on its own status except the lag value supplied from the server, which reveals little information about application usage compared to the centralized implementation. The procedures are summarized in Algorithm 2 with a computational complexity of $O(1)$ at each user and communication overhead of $O(n)$ at the server.

B. Illustration of Control Decisions

In the objective of Eq. (21), $H(t) \sum_{i=1}^{n} g_i(t, t + \tau)$ can be viewed as a penalty term when there are backlogs in the virtual queue. When there is no backlog ($Q(t), H(t) = 0$), we only have the first term in Eq. (21) so the control decision is to always set the device to idle. This matches with the intuition to wait for better co-running opportunities.

### Table I: List of important notations of device $i$.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_i$, $P_i^d$</td>
<td>Average power consumption of training/application co-running, and idling.</td>
</tr>
<tr>
<td>$g_i(t, t + \tau)$, $G(t, t + \tau)$</td>
<td>Gradient gap between time $t$ and $t + \tau$ and the sum of gradient gaps from all the devices.</td>
</tr>
<tr>
<td>$\alpha(t), s(t)$</td>
<td>Control decision {“schedule”, “idle”}, application status {“app”, “no app”}.</td>
</tr>
</tbody>
</table>

### Algorithm 2: Distributed Online Scheduling Algorithm

1. **Input:** Queue backlogs $Q(t)$ and $H(t)$, control parameter $V$, and action space $\Omega$, learning rate $\eta$, momentum vector $v$.
2. **Output:** Scheduling decisions $\forall i$.
3. for $i = 1$ to $n$ do
   4. Send duration $d_i$ to the server, and receives lag $l_{d_i}$ from the server.
   5. Estimate $g_i(t, t + \tau_i)$ with Eq. (4).
   6. $\alpha_i(t) \leftarrow \arg \min_{\alpha_i(t)} \{ V P_i^b(t) - Q(t) b_i(t) + H(t) g_i(t, t + \tau_i). \}
   7. Inform control decision $\alpha_i(t)$ to server.
8. Server: Update $Q(t), H(t)$ according to Eqs. (15) and (16) respectively according to $\alpha_i(t)$.

### No Staleness from the Virtual Queue

There could be cases that there are queue backlogs in $Q(t)$, but for the virtual queue $H(t)$, the cumulative gradient gap has not exceeded the bound $L_0$, i.e., $H(t - 1) + \sum_{i=1}^{n} g_i(t - 1, t + \tau - 1) \leq L_0$. Hence, $H(t) \sum_{i=1}^{n} g_i(t, t + \tau) = 0$ and we derive the decision of,

$$\alpha_i(t) = \arg \min_{\alpha_i(t)} \left\{ \right.$$

$$\left. \begin{array}{l}
(V P_i^b \cdot Q(t) - V P_i^n t), s(t) = \text{‘app’} \\
(V P_i^b \cdot Q(t) - V P_i^n t), s(t) = \text{‘no app’}
\end{array} \right\}$$

(22)

The decision can be made by simply observing $Q(t)$; for $s(t) = \text{‘app’}$, the decision is to co-run if $Q(t) \geq V t(P_i^b - P_i^n)$; otherwise, the decision is idling. Similarly, for $s(t) = \text{‘no app’}$, the decision is to execute as a background process when $Q(t) \geq V t(P_i^b - P_i^n)$ or set to idle otherwise. As a result, the controller would wait until the queue length reaches a certain level.

### With Gradient Staleness

When $H(t) g(t) > 0$, the penalty term $H(t) g(t)$ is active so the control decision accounts for possible staleness.

$$\alpha_i(t) = \arg \min_{\alpha_i(t)} \left\{ \right.$$

$$\left. \begin{array}{l}
(V P_i^b \cdot Q(t) + H(t) \left\| \eta \frac{1 - \beta^\tau}{1 - \beta} v_i \right\|_2, V P_i^n + H(t) g_i(t - 1, t + \tau - 1) \right) , s(t) = \text{‘app’} \\
(V P_i^b \cdot Q(t) + H(t) \left\| \eta \frac{1 - \beta^\tau}{1 - \beta} v_i \right\|_2, V P_i^n + H(t) g_i(t - 1, t + \tau - 1) \right) , s(t) = \text{‘no app’}
\end{array} \right\}$$

(23)

The relevant control decisions can be made by observing $Q(t), H(t)$ and compute the rest of the values in Eq. (23).
C. Optimality Analysis

The optimality of the problem is derived in Theorem 1.

Theorem 1. Let $L(\Theta(t))$ defined by Eq. (17) and $L(\Theta(0)) = 0$, $P^*$ is the optimal power consumption. For constants $B, V \geq 0$, the queues of $\Theta(t)$ are mean rate stable and the time-averaged power consumption and queue backlogs are bounded by:

$$
\limsup_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{P(t)\} \leq \frac{B}{V} + P^* \tag{24}
$$

$$
\limsup_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\Theta(t)\} \leq \frac{B}{\epsilon_1} + \frac{V(P^* - P)}{\epsilon_1} \tag{25}
$$

The proofs are detailed in Appendix X-B. The performance bounds Eqs. (24), (25) demonstrate an $[O(1/V), O(V)]$ energy-staleness trade-off: by arbitrarily increasing $V$, we can make $\frac{B}{V} \to 0$ and the time-averaged power consumption close to the optimal value, whereas the staleness grows linearly with $V$.

VI. SYSTEM IMPLEMENTATION

To conduct neural net training on Android, we adopt a Java-based Deep Learning framework called DL4J [22], which provides seamless integration with the Android OS. The backend neural computations are supported by OpenBLAS cross-compiled for the ARM platforms. We pre-load the CIFAR10 dataset [33] into the flash storage of the phone and retrieve a compiled Java class file into the phone. We use the CIFAR10 dataset for training only, and the neural network is trained on the ARM devices.

A. Energy Measurement

First, we measure the energy consumptions of different control decisions as shown in Table II: training only (1st row - considering training also as an app), application only (1st col.) and co-running (2nd col.). To mitigate the chances of breaking the devices while removing the battery and screen connectors, we use a combination of software profilers: Trepn [23] and Snapdragon Profiler [24] with the Monsoon Power Monitor [25]. Trepn is an old version for the last generation of Snapdragon chipsets (Nexus6/6P) and the newer version of Snapdragon Profiler supports the newer architectures of Pixel 2. For non-Snapdragon chipsets (Hikey970), we directly power the development board with 12V DC input from the Monsoon Power Monitor.

We measure the system-wide energy consumption from the device which includes all the system background threads. To reduce the variances, we disable all irrelevant applications that might have processes lingering in the background. We choose a number of 8 popular applications that users usually spend considerable time. The percentage of energy saving is calculated as, $1 - \frac{P_{st}}{P_{st} + P_{co}}$ (notations from Eq. (10)).

We notice that the newer generations of devices offer much higher energy saving ratio across all the applications (30-50%) and a slightly increased execution time due to contentions of memory bandwidth. However, for the older chipsets such as Nexus 6 with four homogeneous cores, co-running only offers marginal energy improvements depending on the application. Some applications even result an energy surge due to contentions of cache resources, which further leads to CPU throttling and elongated training time. In these cases, the online controller is expected to avoid co-running.

B. Simulation Evaluation

**Evaluation Settings.** We set the probability of application arrival to 0.001 in each time slot, i.e., an average of 1 app
arrival for every 1000s. The application is chosen uniformly random from the 8 representative applications with the running time measured in Table II. We set the number of users to 25 (equal partition of the CIFAR10 dataset) and each user randomly picks a device from the testbed. The total training time is set to 3 hours and each time slot is 1s. We compare the online algorithm against Sync-SGD [2], offline scheduling (Knapsack Problem) and the fixed policy of immediate scheduling, which runs the background training immediately when a device is available regardless of the application arrivals. We set the look-ahead time window of the application arrival. In contrast, with the relaxed staleness bound $L_b = 1000$, the Knapsack offline solution acts almost equivalently to a greedy scheme that is always waiting for co-running opportunities, thus incurs the minimum energy consumption. Reducing the value of $L_b$ would elevate the horizontal line of the offline solution. The online optimization evolves in the space in between: as $V$ grows, the energy consumption quickly drops and slowly approaches the offline solution around 200KJ, with a diminishing marginal gain when $V$ becomes large. The energy consumption stabilizes within an approximation factor of 1.14 to the offline solution and 66% energy savings compared to immediate scheduling and 63% compared to Sync-SGD [2]. Adjusting the staleness bound $L_b$ implies different levels of tolerance to staleness. With a larger $L_b$, more devices are put into the idling mode waiting for applications, thus the energy consumption is lower.

**Comparison of Energy Consumption.** Fig. 4(a) compares the energy consumption of different scheduling policies. Immediate scheduling serves as an upper bound of energy consumption as it quickly turns on training regardless of the application arrival. In contrast, with the relaxed staleness bound $L_b = 1000$, the Knapsack offline solution acts almost equivalently to a greedy scheme that is always waiting for co-running opportunities, thus incurs the minimum energy consumption. Reducing the value of $L_b$ would elevate the horizontal line of the offline solution. The online optimization evolves in the space in between: as $V$ grows, the energy consumption quickly drops and slowly approaches the offline solution around 200KJ, with a diminishing marginal gain when $V$ becomes large. The energy consumption stabilizes within an approximation factor of 1.14 to the offline solution and 66% energy savings compared to immediate scheduling and 63% compared to Sync-SGD [2]. Adjusting the staleness bound $L_b$ implies different levels of tolerance to staleness. With a larger $L_b$, more devices are put into the idling mode waiting for applications, thus the energy consumption is lower.

**Comparison of Convergence speed.** Fig. 5(a) compares the convergence speed of different schemes. Online scheduling converges quickly, while offline scheduling converges slowly. Immediate scheduling converges immediately when a device is available regardless of the application arrival. In contrast, with the relaxed staleness bound $L_b = 1000$, the Knapsack offline solution acts almost equivalently to a greedy scheme that is always waiting for co-running opportunities, thus incurs the minimum energy consumption. Reducing the value of $L_b$ would elevate the horizontal line of the offline solution. The online optimization evolves in the space in between: as $V$ grows, the energy consumption quickly drops and slowly approaches the offline solution around 200KJ, with a diminishing marginal gain when $V$ becomes large. The energy consumption stabilizes within an approximation factor of 1.14 to the offline solution and 66% energy savings compared to immediate scheduling and 63% compared to Sync-SGD [2]. Adjusting the staleness bound $L_b$ implies different levels of tolerance to staleness. With a larger $L_b$, more devices are put into the idling mode waiting for applications, thus the energy consumption is lower.
the online scheme only lags the immediate scheduling by 1000s, but offers almost 60% energy saving and both schemes ultimately converge to the same range of accuracy. The offline and Sync-SGD fall far behind mainly due to insufficient number of updates from the users. To compare the wall-lock training time more closely, we record the time for different schemes to reach 55%, 50%, 45% and 40% testing accuracy in Fig. 5(c), by varying the random seeds to generate different random permutations of devices and application types. Since the testing accuracy of Sync-SGD has plateaued around 50%, it never reaches 55% during the 3-hour time frame. Similar to the previous results, Sync-SGD and offline scheme both result in the largest convergence time with different accuracy objectives. Immediate scheduling offers the fastest training with much higher cost of energy, especially when the accuracy objectives are lower (0.45, 0.4). The online scheme is capable of achieving a reasonable trade-off between energy consumption and training time.

**Impact of Application Arrival**. Our strategy relies on the intensity of application usage for energy saving. We further evaluate the impact of different application arrival rates varying from $10^{-4}$ to 0.2 per time slot, especially when the applications are scarce. Fig. 6(a) shows the application arrival rate vs. energy consumptions. With more running applications, the general energy consumption follows an increasing trend for all three schemes. Immediate scheduling is independent of application arrivals and the energy saving comes from the coincident co-running. In contrast, the online scheme is able to utilize the application arrival more wisely as we can see the initial gap from immediate scheduling is large. As the application rate rises, co-running quickly saturates and the online scheme has degraded into the immediate scheme. Because the offline scheme foresees all the co-running opportunities, it is able to achieve the lowest energy consumption when applications are scarce but will aggressively schedule with the applications when the arrival rate increases. Due to the random arrivals, the energy consumption has more variance with a larger arrival rate. As application usage depends on a variety of contextual cues such as time and location [29], it is highly possible that there is few application usage. Fig. 6(b) shows the impact on testing accuracy when applications are scarce. We can see that there is no noticeable accuracy degradation for the online scheme. Once the cost of the queue backlogs increases, the online scheme is able to switch back into the immediate scheme to clear the queue congestions. Thus, the offline scheme may offer better energy efficiency for different application rates, but the control decisions generate a negative feedback on
the upper level convergence and testing accuracy when the applications are scarce. The online scheme provides more flexibility to adapt different application arrivals.

**Energy Overhead.** The online scheme examines Eq. (21), which involves lightweight computation on the little cores. The energy overhead is shown in Table III, which is below 10% in each time slot. To reduce the overhead, we can adjust the scheduling granularity. E.g., instead of making a decision in each time slot, we can enlarge the decision intervals, whereas this might miss co-running opportunities if the interval is larger than the application execution time. Due to space limit, we will demonstrate such trade-off in an extended version of this work.

### Table III: Energy overhead of online optimization (W).

<table>
<thead>
<tr>
<th>Power(idle)</th>
<th>Power(comp.)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.238</td>
<td>0.486</td>
<td>3.0%</td>
</tr>
<tr>
<td>0.245</td>
<td>0.525</td>
<td>7.4%</td>
</tr>
<tr>
<td>0.736</td>
<td>6.3%</td>
<td></td>
</tr>
</tbody>
</table>

**VIII. CONCLUSION AND FUTURE WORK**

In this paper, we combine ASync-SGD and application co-running to minimize energy consumptions of federated tasks on mobile devices. We motivate this work by real measurements and illustrate the fundamentals of energy saving. Then we develop the offline and online schemes to explore the energy-staleness trade-off with low computational overhead. Our extensive evaluation demonstrates that the online optimization achieves over 60% energy saving compared to the benchmarks, and only 15% away from the offline solution.

The proposed mechanism can adapt to different diurnal and nocturnal application usage patterns by taking advantage of the common temporal activities from the users, while keeping the devices in low power state during the rest of the time. Though we only demonstrate the convergence empirically, in principle, the theoretical convergence is guaranteed given a bounded delay of gradients [36]. The Lyapunov framework manages the delay from the virtual queue bounded by the gradient staleness $L_b$ in Eq. (14). We defer the rigorous theoretical proofs to the future works.

**IX. ACKNOWLEDGEMENT**

We are grateful to the anonymous reviewers for their insights and detailed comments, and the support from the U.S. National Science Foundation under grant number 2152580 and 2007386.

**X. APPENDIX**

### A. Proofs of Lemma 2

Applying Eq. (17) to Eq. (18), we have,

$$\Theta(t) = E\{L(\Theta(t+1)) - L(\Theta(t))|\Theta(t)\}$$

$$= \frac{1}{2}E\{\Theta(t+1)^2 - \Theta(t)^2\}$$

$$= \frac{1}{2}E\{Q(t+1)^2 - Q(t)^2 + H(t+1)^2 - H(t)^2\}$$

Since $\max^2\{x, 0\} \leq x^2$, from Eq. (15) and Eq. (16) we have,

$$Q^2(t+1) + H^2(t+1) \leq Q^2(t) + (A(t) - b(t))^2 + 2Q(t)(A(t) - b(t)) + H^2(t) + G(t)^2 + 2H(t)G(t) + L_b^2$$

From (26),

$$\Theta(t) \leq B + E\{Q(t)(A(t) - b(t))|Q(t)\} + E\{H(t)G(t)|H(t)\}$$

where $B = \frac{1}{2}(A_{max}^2 + B_{max}^2 + G_{max}^2 + L_b^2)$. $A_{max}$, $B_{max}$ and $G_{max}$ are the maximum arrival, service rate and gradient gap in the system. Thus, Eq. (28) completes the proof of Lemma 2.

### B. Proofs of Theorem 1

For the optimal decision $\alpha^*(t)$ that can stabilize the queue,

$$E\{P(\alpha^*(t))\} = P^*$$. (29)

Since we can adjust the control decisions, there must exist $\epsilon_1, \epsilon_2 > 0$ so the difference between the service and arrival rates of the actual and virtual queues are larger than $\epsilon_1, \epsilon_2$ respectively (considering $L_b$ as the fixed service rate of the virtual queue $H(t)$):

$$E\{b_i(t) - A_i(t)|Q(t)\} > \epsilon_1$$

$$E\{L_b - g_i(t, t + \tau)|H(t)\} > \epsilon_2$$

### Plugging Eqs. (30) and (31) into Eq. (20),

$$\Delta(\Theta(t)) + V\{P(t)|\Theta(t)\} \leq B + VP^* - \epsilon_1E\{Q(t)\} - \epsilon_2E\{H(t)\}$$

### Taking the summation over $t \in \{0, \cdots, T - 1\}$,

$$\sum_{t=0}^{T-1} \Delta(\Theta(t)) + \sum_{t=0}^{T-1} V\{P(t)|\Theta(t)\} \leq T(B + VP^*)$$

$$- \sum_{t=0}^{T-1} (\epsilon_1E\{Q(t)\} + \epsilon_2E\{H(t)\})$$

### Plugging Eq. (18) into Eq. (33), and dividing both sides by $T \cdot V$,

$$\frac{E\{L(\Theta(T - 1)) - L(\Theta(0))\}}{TV} + \frac{1}{T} \sum_{t=0}^{T-1} E\{P(t)\} \leq \frac{B}{V}$$

$$+ P^* - \frac{\epsilon_1}{TV} E\{Q(t)\} - \frac{\epsilon_2}{TV} E\{H(t)\}$$

### Since $L(\Theta(0)) = 0$, when $T \to \infty$, we can simplify Eq. (34) as,

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} E\{P(t)\} \leq \frac{B}{V} + P^*$$

### The time averaged queue length can be derived by dividing $\epsilon T$,

$$\frac{1}{T} \sum_{t=0}^{T-1} E\{\Theta(t)\} \leq B + V(P^* - \frac{1}{T} \sum_{t=0}^{T-1} E\{P(t)\})$$

$$- \frac{(\sum_{t=0}^{T-1} \epsilon_1E\{Q(t)\} + \epsilon_2E\{H(t)\})}{\epsilon T} + \frac{E\{L(\Theta(0))\}}{\epsilon T}$$

(36)
Taking the limits of $T \to \infty$,
\[
\Theta = \limsup_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\Theta(t)\} \leq \frac{B}{\epsilon} + \frac{V(P^* - \overline{P})}{\epsilon}
\]  

(37)

REFERENCES


